**EXPANDED PAPER**

**OPREA DARIUS-EMANUEL**

**Artificial Intelligence and Vision, I, Feb 2025**

**Motivation for Expanding the Paper**

The proposed paper, **Multi-Agent Architecture for Information Retrieval and Intelligent Monitoring by UAVs in Known Environments Affected by Catastrophes -** *D. Vallejo,* is studying catastrophes, whether natural or man-made, that require rapid environmental analysis for effective crisis management and alert notification.

UAVs play a crucial role in supporting rescue teams by enabling intelligent monitoring systems. This article proposes a **multi-agent architecture** for deploying intelligent agents that assess affected areas and aid *decision-making*. These agents analyze predefined critical points using a formal normality analysis model and communicate to assess overall risk. A case study demonstrates the system’s application in monitoring the spread of toxic smoke following an industrial earthquake.

To evacuate workers as soon as possible, the researchers monitored the exit/entering roads of the industrial complex using the multi-agent architecture.

Expanding the paperwork is motivated because their solution could lead to congestion. And if the system recommends one route of evading for everyone, then that specific road becomes congested.

**Key Notes of the Solution Proposed**

I managed to think of a solution that can work for most of the situations in such a big complex, an industrial one, for example. An industrial complex has multiple buildings that are far away from one another. By selecting a meeting point that can be a big yard, a park, basically, a point where everyone should meet in case of an emergency, away from any building or structure.

Trying to get the **shortest path** from any building to the meeting point is considered the main goal of the expanded version of the paper.

Also, I think that any route can get “**canceled**” because of *congestion* or by *any incident* that can cause **obstructions** on the road. By doing so, the system can *recalculate* the path of a missing route or a route with different cost – that was updated because of the weighting cost received from the “blackboard” – the shared layer where any drone interacts with it and centralizes data.

**The Tools I Used**

To build the application that can simulate the building, I used the **Vite** build tool used for the development of web projects. **Javascript** project using **vis-network** library for visualising the graph – *the schematic representation of the industrial complex*. To represent the matrix of adjacency, I used **handsontable** library.

**Experimental Setup**

In the first place, I was thinking of how to represent the industrial complex and its inner roads more programmatically.

I ended up using a graph. A graph where every route has a cost – the total minutes a normal human being can take from one place to another. I represented this graph using a matrix of adjacency. This matrix was “**delivered**” by Blackboard. If the blackboard is updating the weight of cost for any route, then our system should recalculate the graph and the minimum cost.

What is the minimum cost approach?

The idea is to generate an SPT – Shortest Path Tree with a given source as a root – *I selected node 0 to be the root, meaning the “****meeting point****”*. Maintain an Adjacency Matrix with two sets.

* one set contains nodes included in the shortest-path tree,
* other set includes nodes not yet included in the shortest-path tree.

At every step of the algorithm, find a node that is in the other set (set not yet included) and has a minimum distance from the source.

* Why Dijkstra’s algorithm?

My graph has a limited number of nodes, no more than 100, and in most cases, no more than 10, and the algorithm fits perfectly for smaller graphs like this. For such a small graph, even the Bellman-Ford method would be fine, though it’s slower in big-O terms. The complexity of Dijkstra can be as low as **O((e + n)logn)**, where e is the *number of edges* and n is the *number of nodes*.

* Dealing with dynamic updates – congestion re-weighting

If the cost on edges can change frequently – for us is not that frequent, but it still can happen that some routes will become more congested than normal, so the cost should be recalculated – then we do have a dynamic graph. Re-running Dijkstra after each change. In our case, this is perfect. For larger graphs or much more frequent updates, we should consider an incremental/dynamic shortest-path technique that can be much efficient.

**Results and utility of the app**

A grid of numbers with black numbers

AI-generated content may be incorrect.

Figure 1 webApp - Adjacency Matrix

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 2 code - Adjacency Matrix

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 3 Selecting a starting node

*You can select any node to see the actual path that you should follow to get from one node to the meeting point, Node 0.*

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4 Information section - COST and PATH

A diagram of a number of objects

AI-generated content may be incorrect.

Figure 5 Graph and Highlighted Path for Node 5

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6 webApp landing Page

A screenshot of a graph

AI-generated content may be incorrect.

Figure 7 webApp working

My solution works because of using the Dijkstra Algorithm to get the path with the minimum cost. It is also good and easy for anyone to use.

You can find the application code and how to run it locally at the Github listed below.

**Conclusions and future work**

The proposed system demonstrates an efficient and practical approach for emergency evacuation scenarios. This can fit in industrial complexes, leveraging **multi-agent architecture** combined with ***graph-based algorithms*** and ***modern web development tools***.

The application successfully utilized Dijkstra’s Algorithm to calculate the shortest paths from any building or structure to a predefined meeting point. This algorithm was chosen due to its efficiency for small-to-medium graphs.

The system is proposing to recalculate paths dynamically in response to changes in edge weights due to **congestion or obstruction**. This information is delivered from a multi-agent system via **Blackboard**. The use of the blackboard system centralizes data collection and decision-making, enabling UAV agents to share updates on congestions or incidents.

Future work that I could do to this project and paper itself is to create and integrate **real-time APIs** that *fetch* the *adjacency matrix* **dynamically**, reflecting real-time conditions. Automatically trigger path recalculations whenever the matrix is updated.

The scalability for larger graphs would be to change the algorithm to **A\***, for example, which enhances the power of dynamic shortest path techniques to handle graphs that change more often.

**Bibliography**

1. SPA: <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>
2. Vallejo, D., Albusac, J., Glez-Morcillo, C., & Castro-Schez, J. J. (2013). Multi-agent architecture for information retrieval and intelligent monitoring by UAVs in known environments affected by catastrophes. IEEE Intelligent Systems
3. Vite: <https://vite.dev/>
4. Vis-network: <https://visjs.github.io/vis-network/docs/network/>
5. handsontable: <https://handsontable.com/>
6. GitHub repository: